

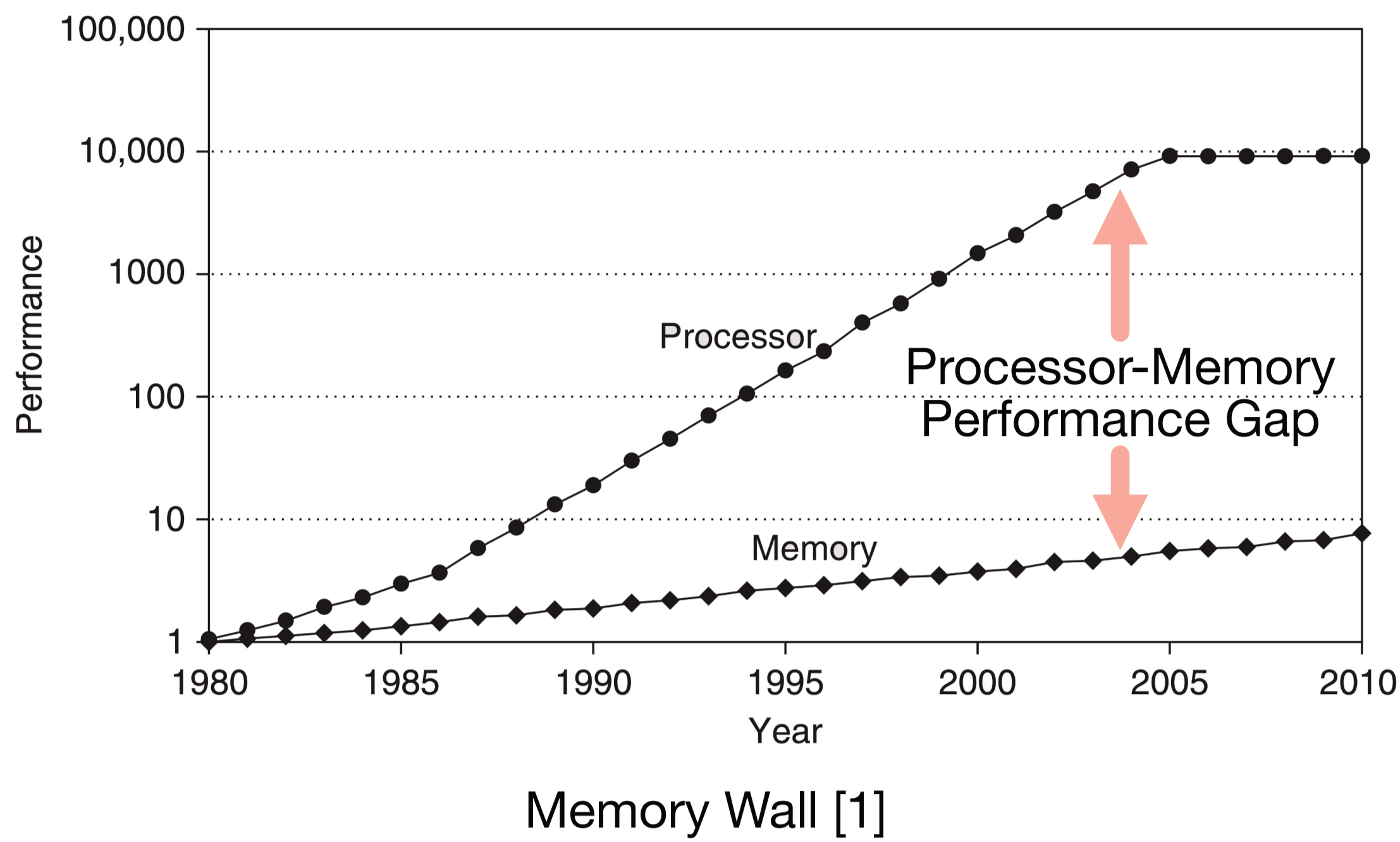
# PyCacheGen: A Highly Configurable Open-Source Generator for Synthesizable Caches

Richard Müller\*, Konstantin Lübeck\*, Michael Kuhn  
Paul Palomero Bernardo, and Oliver Bringmann

\*These authors contributed equally to this work.

✉ konstantin.luebeck@uni-tuebingen.de • embedded.uni-tuebingen.de

## Problem



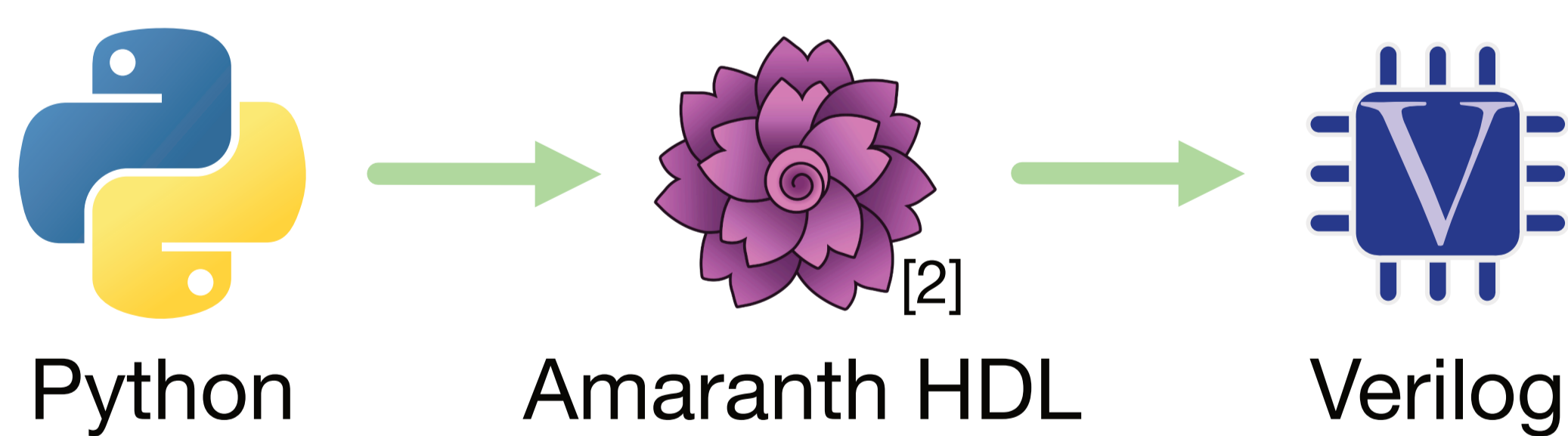
To bridge this gap, a highly configurable cache generator is needed to find the best cache configuration for a given set of applications to satisfy performance, power, and area requirements.

## PyCacheGen

```
from pycachegen import *
from amaranth.back import verilog

# configure cache hierarchy
cache_wrapper = CacheWrapper(
    num_ports=1,
    address_width=25,
    cache_configs=[
        CacheConfig(
            data_width=32, num_ways=2,
            num_sets=1, block_size=2,
            replacement_policy=ReplacementPolicies.PLRU_TREE,
            write_policy=WritePolicies.WRITE_BACK,
            write_allocation=True, write_buffer_size=4
        ),
        CacheConfig(
            data_width=32, num_ways=2,
            num_sets=2, block_size=4,
            replacement_policy=ReplacementPolicies.FIFO,
            write_policy=WritePolicies.WRITE_BACK,
            write_allocation=True, write_buffer_size=8
        )
    ],
    main_memory_data_width=128,
    create_main_memory=False
)

# generate verilog code
with open("cache_wrapper.v", "w") as f:
    f.write(
        verilog.convert(cache_wrapper, name="CacheWrapper")
    )
```

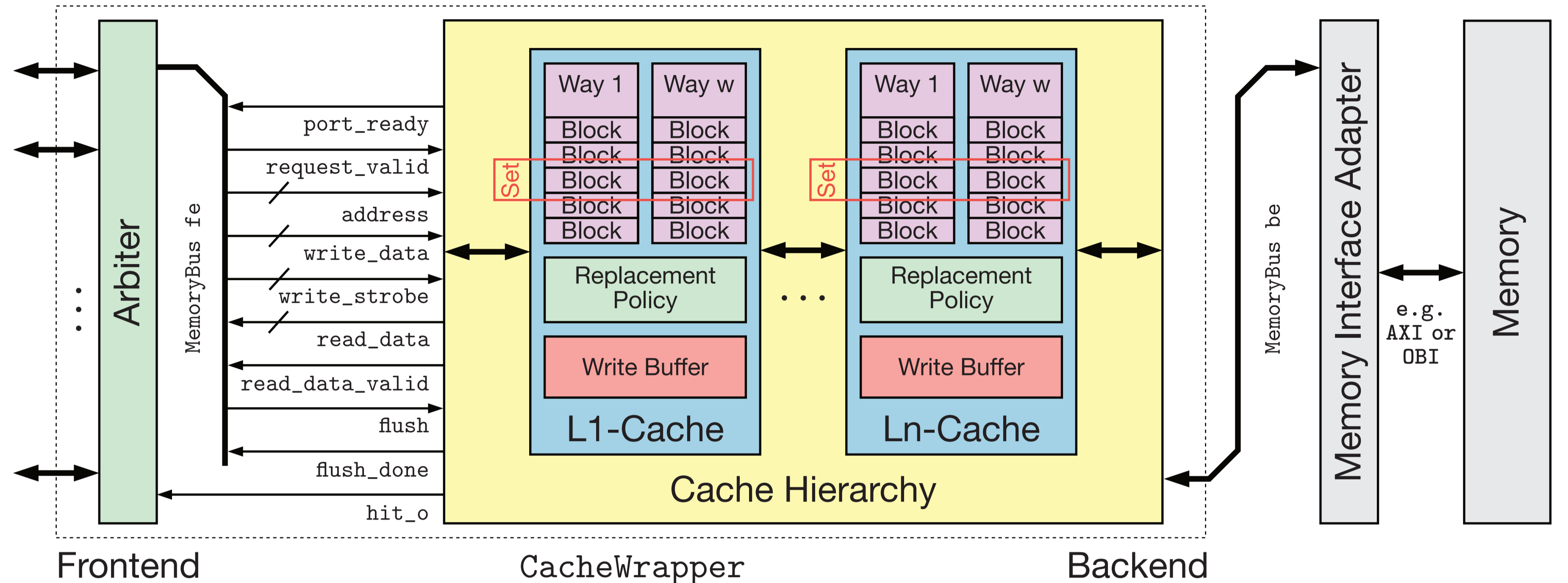


## Star us on GitHub



<https://github.com/ekut-es/pycachegen>

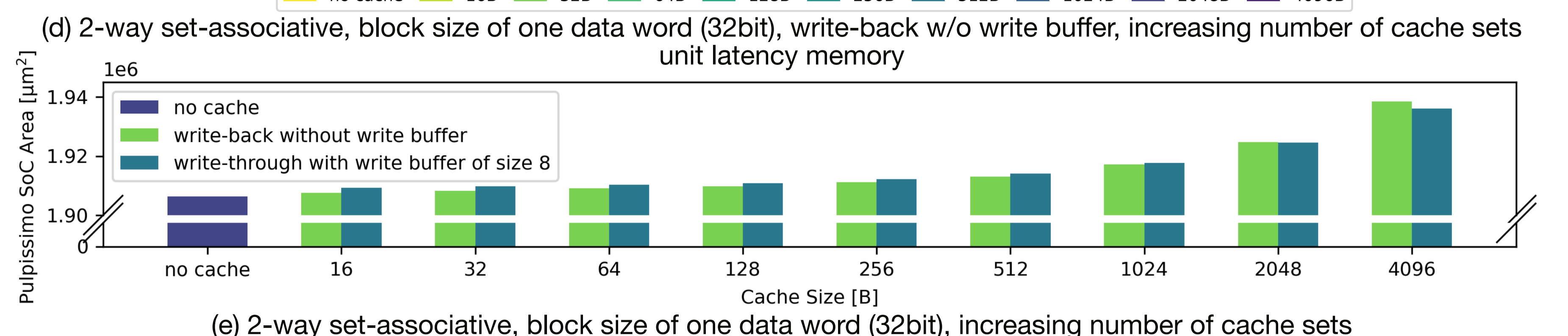
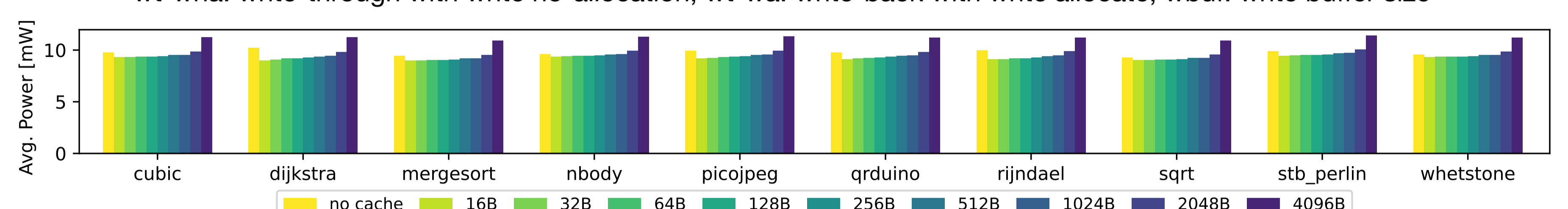
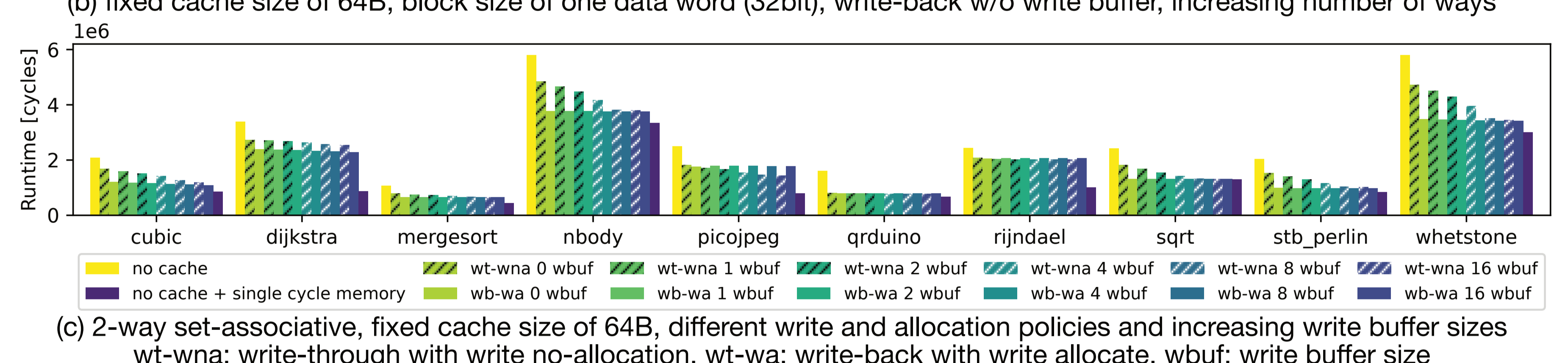
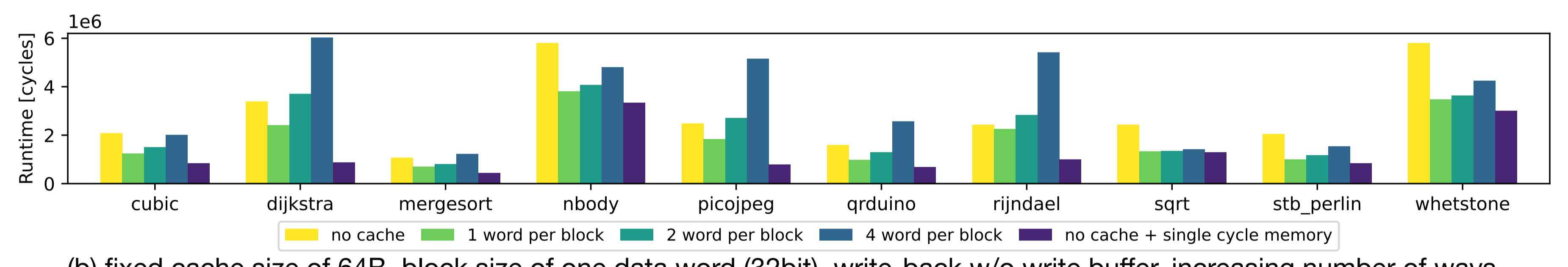
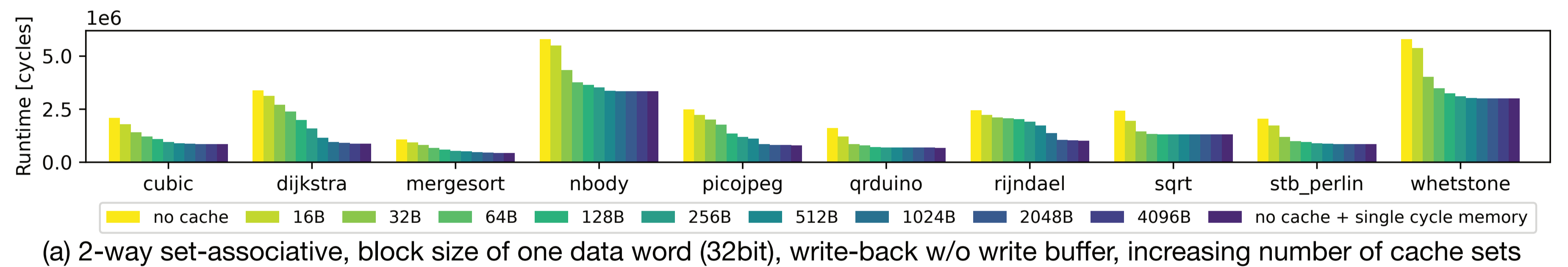
## Features



- Fully associative
- Set-associative
- Direct mapped
- Write-through
- Write-back
- Write allocate
- Write no-allocate
- Write buffer
- Configurable data width
- Configurable block size
- Single cycle hit latency
- Different replacement policies
- Multiple request ports
- Flushing
- Multiple cache levels

## Results

We integrated caches of different sizes into the PULPissimo SoC [3] between the memory and the RISC-V core. RTL simulations were used to gather runtime results. For power, energy, and area data, the PULPissimo SoC was synthesized using GlobalFoundries' 22FDX+ technology at a clock frequency of 200 MHz, utilizing standard cells and low-leakage memory macros from Synopsys. We selected ten benchmarks from the BEEBs suite [4]. For (a), (b), (c) the memory latency was increased.



## References

- [1] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Fifth Edition. Heidelberg: Elsevier, Morgan Kaufmann, 2012.
- [2] Amaranth HDL, 2025. URL: <https://github.com/amaranth-lang/amaranth>.
- [3] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, "Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX," in 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), Burlingame, CA, USA: IEEE, Oct. 2018, pp. 1-3
- [4] J. Pallister, S. Hollis, and J. Bennett, "BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms," Sept. 28, 2013, arXiv: arXiv:1308.5174

